THE HONG KONG POLYTECHNIC UNIVERSITY
HONG KONG COMMUNITY COLLEGE

Subject Title	: Data Structures	**Subject Code**	: CCN2239
Session	: Semester One, 2016/17	**Time**	: 09:30 – 12:30
Date	: 11 December 2016	**Time Allowed**	: 3 Hours
Subject Examiner(s)	: Dr Pat Chan Dr Ken Tsang		

This question paper has a total of **TWENTY-ONE** pages (including this covering page).

Instructions to Candidates:

1. There are THREE sections in this paper.
   - Section A (30%) – Multiple-choice Questions. Answer ALL questions in this section on the multiple-choice answer sheet provided. Each question carries 1 mark. Choose the BEST option for each question.
   - Section B (50%) – Short Questions. Answer any FIVE out of the SIX questions in this section in the answer book provided. Each question carries 10 marks. If you answer more than five questions, only the first five attempted questions will be marked. Indicate in your answer book clearly which five questions you are attempting.
   - Section C (20%) – Long Questions. Answer any ONE out of the TWO questions in this section in the answer book provided. Each question carries 20 marks. If you answer more than one question, only the first one attempted question will be marked. Indicate in your answer book clearly which one question you are attempting.
2. Candidates are NOT allowed to retain the multiple-choice answer sheet, the answer book and the examination question paper.
3. Show all your work clearly and neatly. Marks will be deducted for untidy work.
4. Reasonable steps should be shown.
5. All programming code must be written in Java programming language.

Authorised Materials:

	YES	NO
CALCULATOR	[ ]	[✓]
SPECIFICALLY PERMITTED ITEMS	[ ]	[✓]

DO NOT TURN OVER THE PAGE UNTIL YOU ARE TOLD TO DO SO

Section B (50%) – Short Questions

Answer any FIVE out of the SIX questions in this section in the answer book provided. Each question carries 10 marks. If you answer more than five questions, only the first five attempted questions will be marked. Indicate in your answer book clearly which five questions you are attempting.

Question B1

(a) For the key sequence given below, determine the binary search tree obtained when the keys are inserted one-by-one in the given order, starting from an initially empty tree:

12, 8, 9, 18, 15, 5, 10, 3, 25

(4 marks)

(b) Consider the binary search tree given in Figure 5 below.

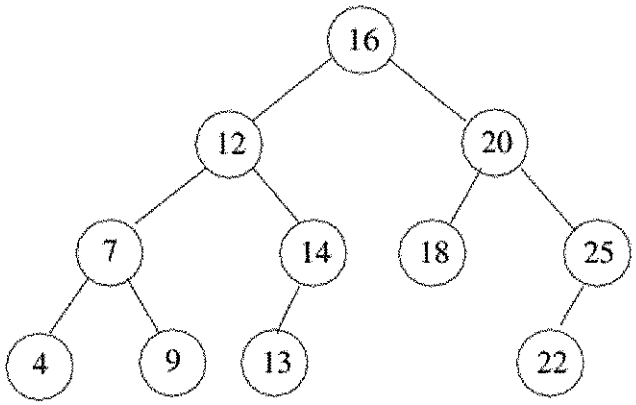

Figure 5

(i) Draw the binary search tree after removing the node with key 9 from Figure 5. (1 mark)

(ii) Draw the binary search tree after removing the node with key 25 from Figure 5. (2 marks)

(iii) Draw the binary search tree after removing the node with key 12 from Figure 5. (3 marks)

Question B2

Write a method called `bidiBubbleSort` that is a bi-directional version of bubble sort on an array of integers with early termination. It makes alternating sweeps over the array, first going from the front to the end, then from the end to the front, and so on. When sweeping from front to end, it swaps forward an element that is larger than the element after it; when sweeping from end to front, it swaps backward an element that is smaller than the element before it. You may assume that the array is not null.

The following diagram shows an execution of this algorithm over a given array:

```
{16, 21, 45, 3, 11, 53, 8, 26, 49}
--> sweep right
16 21 3 11 45 8 26 49 53
<-- sweep left
3 16 21 8 11 45 26 49 53
--> sweep right
3 16 8 11 21 26 45 49 53
<-- sweep left
3 8 16 11 21 26 45 49 53
--> sweep right
3 8 11 16 21 26 45 49 53
<-- sweep left
end
```

(10 marks)

Question B3

(a) Suppose that A is an array storing n identical integer values. Which sorting algorithm has the smallest running time when this array A is given as input? Explain your answer. (3 marks)

(b) Instead of sorting the entire data set, you only need the k smallest elements where k is an input to the algorithm but is likely to be much smaller than the size of the entire data set. Which sorting algorithm is the most suitable for this situation? Explain your answer. (3 marks)

(c) Sort the following sequence of keys using merge sort and write down clearly the sequences in each pass. (4 marks)

`69, 72, 3, 81, 106, 27, 35, 48, 52`

Question B4

Consider the following `Node` and `List` classes:

```
class Node {
   int item; // element in the node
   Node next; // next is the pointing at the next node
   Node(int item) {
      this.item = item;
      this.next = null;
   }
   Node(int item, Node next) {
      this.item = item;
      this.next = next;
   }
}

class List{
    private Node head; // point to first node in the list
}
```

Consider the linked list of integer represented by the following diagram in Figure 6:

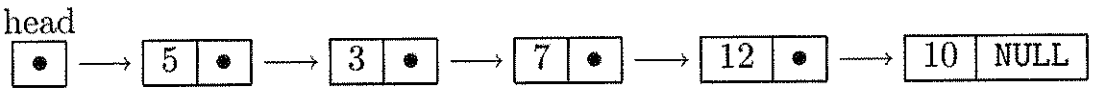

Figure 6

(a) Draw a diagram of the above list after the following lines of code have executed:

```
Node prev = head.next;
Node nodeToInsert = new Node(4);
nodeToInsert.next = prev.next;
prev.next = nodeToInsert;
```

(4 marks)

(b) Assume that the code represented above in part (a) in this question has executed. What is the value of `prev.item`? (2 marks)

(c) In addition to the code in part (a) in this question, assume the following code executes. Draw a diagram of the list after this code executes as well. (4 marks)

```
prev = prev.next;
prev = prev.next;
Node curr = prev.next;
prev.next = curr.next;
curr = null;
```

Question B5

Consider a postfix calculator that uses a stack to evaluate a postfix expression.

(a) Given the posfix expression below, what is the corresponding infix expression of the arithmetic calculation?

```
3 5 4 - / 8 *
```

(2 marks)

(b) Following part (a) in this question above, write down the numbers inside the stack when an operand or an operator in the expression is entered into the calculator in each round. Indicate clearly in your answer which side is the bottom of the stack. (3 marks)

(c) Consider the Java program segment below that implements a simple postfix calculator. By using the push and pop methods in the Java class `ArrayStack`, fill in the codes for the blanks <1> to <4> to complete the program. (5 marks)

```
ArrayStack s = new ArrayStack();
String expr = "3 5 4 - / 8 *";
for (int i = 0; i < expr.length(); i += 2){
     char c = expr.charAt(i);
     if (Character.isDigit(c))  {
          int op = c - '0';
          // <1>

     }
     else {
          int op1, op2;

          // <2>

          int result = 0;
          switch(c){
          case '+': result = op1 + op2; break;
          case '-': result = op1 - op2; break;
          case '*': result = op1 * op2; break;
          case '/': result = op1 / op2; break;
          }

          // <3>

     }
}
int answer;

// <4>

System.out.println("The answer is " + answer);
```

Question B6

(a) Given the following binary tree in Figure 7:

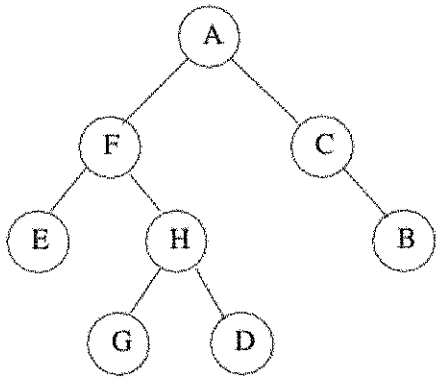

Figure 7

(i) List the sequence of nodes visited by preorder traversal. (2 marks)

(ii) List the sequence of nodes visited by inorder traversal. (2 marks)

(iii) List the sequence of nodes visited by postorder traversal. (2 marks)

(b) Suggest **TWO** differences between a tree and a binary tree. (4 marks)

- End of Section B -

Section C (20%) – Long Questions

Answer any ONE out of the TWO questions in this section in the answer book provided. Each question carries 20 marks. If you answer more than one question, only the first one attempted question will be marked. Indicate in your answer book clearly which one question you are attempting.

Question C1

A *circular buffer* is a fixed-size queue structure in which the rear end of the queue is joined with the front to make a circle. Two pointers are used with a circular buffer for reading and writing. Figure 8 shows the position of these pointers for a circular buffer containing 5 integers.

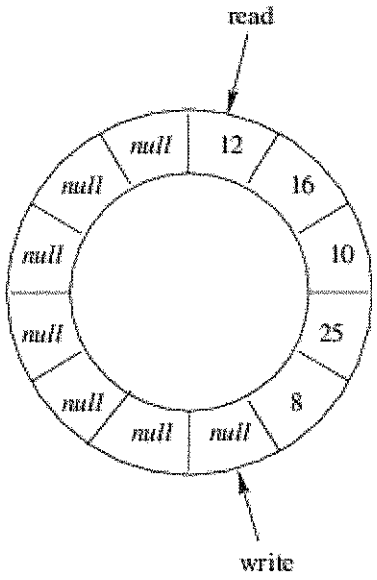

Figure 8

The *read* pointer always points to the next element to read, and the *write* pointer always points to the next empty slot for writing the next element. After an element is read, that element is removed from the buffer and the *read* pointer moves (clockwise) to point to the next slot. After an element is written, it is added into the buffer and the *write* pointer moves (clockwise) to point to the next slot.

When a circular buffer is empty, the *read* and *write* pointer points to the same slot (see Figure 9). When a circular buffer is full, the *write* pointer points to the last empty slot (see Figure 10). Note that such circular buffer is still considered as full, although there is still an empty slot left.

Question C1 (continued)

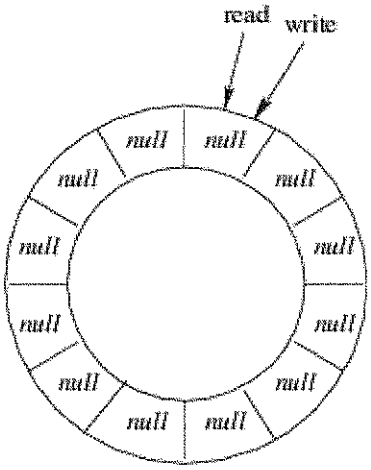

Figure 9

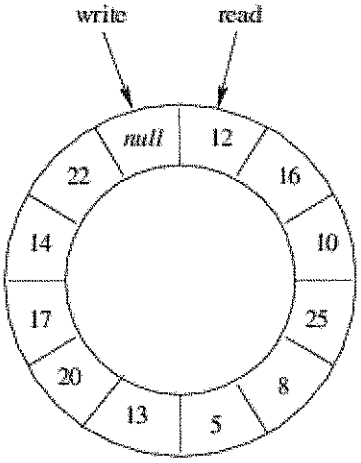

Figure 10

In the above example, the circular buffer can hold at most 11 elements, i.e., the capacity of the circular buffer is 11.

Given the partially-completed Java program `LinkedCircularBuffer` below, you are asked to implement a circular buffer using a linked list of nodes (`ChainNode`). Assume both `LinkedCircularBuffer.java` and `ChainNode.java` are in the same package.

```
// LinkedCircularBuffer.java
public class LinkedCircularBuffer {

     private ChainNode read;
     private ChainNode write;

     public LinkedCircularBuffer(int initialCapacity){

          // <1>: Codes to be completed in part (b)

     }
```

Question C1 (continued)

```
public boolean isEmpty(){

     // <2>: Codes to be completed in part (c)

}

public boolean isFull(){

     // <3>: Codes to be completed in part (d)

}

public Object readElement(){

     // <4>: Codes to be completed in part (e)

}

public void writeElement(Object theObject){

     // <5>: Codes to be completed in part (f)

}

public static void main(String[] args) {

   LinkedCircularBuffer buf = new LinkedCircularBuffer(10);

     for (int i = 1; i <= 5; i++){
          buf.writeElement(new Integer(i));
     }

     System.out.println("Reading 3 elements:");
     for (int i = 1; i <= 3; i++){
          Object e = buf.readElement();
          System.out.print(e.toString() + " ");
     }
     System.out.println();

     for (int i = 1; i <= 4; i++){
          buf.writeElement(new Integer(i * 10));
     }

     System.out.println("Reading 5 elements:");
     for (int i = 1; i <= 5; i++){
          Object e = buf.readElement();
          System.out.print(e.toString() + " ");
     }
     System.out.println();
  } }
```

Question C1 (continued)

```
// ChainNode.java
class ChainNode
{
   // package visible data members
   Object element;
   ChainNode next;

   // package visible constructors
   ChainNode() {}

   ChainNode(Object element)
      {this.element = element;}

   ChainNode(Object element, ChainNode next)
      {this.element = element;
       this.next = next;}
}
```

Sample output

```
Reading 3 elements:
1 2 3
Reading 5 elements:
4 5 10 20 30
```

(a) Consider the circular buffer with 5 integers in Figure 8. Draw the resulting circular buffer after the read and write operations below. Indicate the positions of the *read* and *write* pointers clearly in your answer.

Read, Read, Write 30, Read, Write 20, Write 40, Read, Read (5 marks)

(b) Fill in your codes in blank <1> to complete the constructor method of `LinkedCircularBuffer`. It creates a circular buffer with capacity *initialCapacity* using a linked list of `ChainNodes`. Note that the circular buffer needs *initialCapacity+1* ChainNodes to support a capacity of *initialCapacity*, because the circular buffer is considered full when there is only one empty slot left. (6 marks)

(c) Fill in your codes in blank <2> to complete the method `isEmpty` which returns true if the circular buffer is empty, and returns false if otherwise. (1 mark)

(d) Fill in your codes in blank <3> to complete the method `isFull`, which returns true if the circular buffer is full, and returns false if otherwise. (1 mark)

(e) Fill in your codes in blank <4> to complete the method `readElement`, which removes the element to read in the circular buffer, and returns that element. If the circular buffer is empty, the method returns null. (4 marks)

(f) Fill in your codes in blank <5> to complete the method `writeElement`, which writes the element *theObject* into the circular buffer. If the circular buffer is full, no writing is performed. (3 marks)

Question C2

The Figure 11 shows the linked list structure of the 2 × 4 matrix with the top left entry pointed by `arr1` pointer after executing the following statements:

```
XYNode arr1 = new XYNode(2);
xLink(arr1, 2, 4);
```

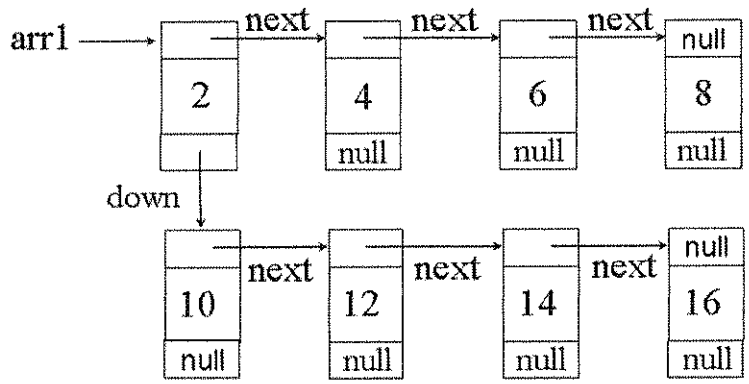

Figure 11

The `XYNode` and `XYNodeTest` class are given in the following Java code:

```
class XYNode {

     // <1>: Codes to be completed in part (a)

}

public class XYNodeTest {

  public static void xLink(XYNode m, int row, int col){

          XYNode row_p, col_p;
          double count = 4.0;
          row_p = m;
          for (int i = 0; i < row; i++) {
               col_p = row_p;
               for (int j = 1; j < col; j++) {
                    col_p.next = new XYNode(count);
                    col_p = col_p.next;
                    count += 2;
               }
               if (i + 1 < row) {
                    row_p.down = new XYNode(count);
                    row_p = row_p.down;
                    count += 2;
               }
          }
   }
public static void yLink(XYNode m, int row, int col){
```

Question C2 (continued)

```
        // <2>: Codes to be completed in part (b)

    }

    public static void DoubleMatix(XYNode m, int row, int col){

        // <3>: Codes to be completed in part (c)

    }

    public static void displayArray(XYNode m) {

        XYNode row_p, col_p;
        row_p = m;
        while (row_p != null) {
             col_p = row_p;
             while (col_p != null) {
                  System.out.print((int) col_p.item + " ");
                  col_p = col_p.next;
             }
             System.out.println();
             row_p = row_p.down;
        }
        System.out.println();
        }

    public static void main(String [] args) {

        XYNode arr1 = new XYNode(2);
        XYNode arr2 = new XYNode(2);
        xLink(arr1, 2, 4);
        yLink(arr1, 2, 4);
        System.out.println("Array arr1 is: ");
        displayArray(arr1);
        xLink(arr2, 4, 2);
        yLink(arr2, 4, 2);
        System.out.println("Array arr2 is: ");
        displayArray(arr2);
        DoubleMatix(arr1, 2, 4);
        System.out.println("Array arr1 after double is: ");
        displayArray(arr1);

        }
    }
```

Question C2 (continued)

(a) Fill in your codes in blank <1> to complete the class `XYNode` such that this class contains **THREE** attributes as the structure shown in Figure 11 and three constructors with the following signatures to initialize the attributes. (5 marks)

```
XYNode(double item);
XYNode(double item, XYNode next);
XYNode(double item, XYNode next, XYNode down);
```

(b) Fill in your codes in blank <2> to complete the `ylink()` method such that this method builds up the `down` pointers for the given *`row` × `col`* matrix with the top left entry pointed by pointer `m`. The Figure 12 shows the linked list structure after executing the following statements:

```
XYNode arr1 = new XYNode(2);
xLink(arr1, 2, 4);
yLink(arr1, 2, 4);
```

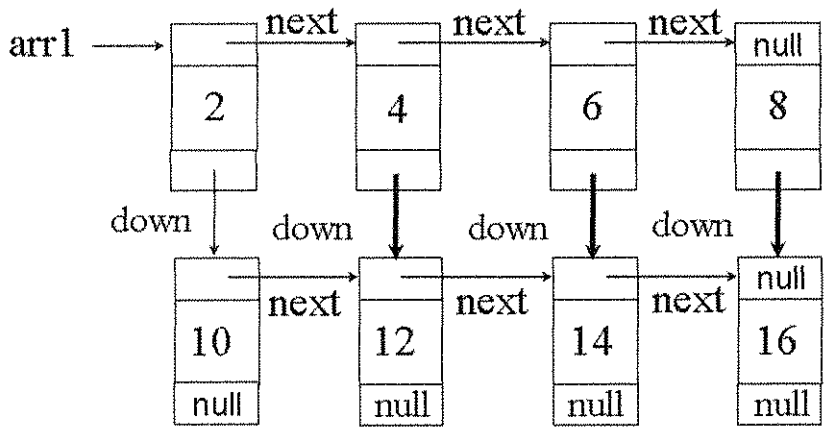

Figure 12

(8 marks)

(c) Fill in your codes in blank <3> to complete the method `DoubleMatrix` such that this method doubles the values of all the numbers stored in the matrix. (4 marks)

(d) What is the output after successfully executing `java XYNodeTest`? (3 marks)

- End of Section C -

- END OF PAPER -